Лабораторная работа № 2 по курсу : Операционные системы

Выполнил студент группы М8О-201Б-21 Кварацхелия Александр.

Условие

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в операционной системе UNIX/LINUX. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или каналы (*pipe*). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Задание

Родительский процесс создает два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами.

Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в *pipe*1 или в *pipe*2 в зависимости от правила фильтрации. Процесс *child*1 и *child*2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод.

Правило фильтрации: с вероятностью 80% строки отправляются в pipe1, иначе в pipe2. Дочерние процессы удаляют все гласные из строк.

Код программы

utils.h

```
#ifndef UTILS_LAB4
#define UTILS_LAB4

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* ReadString(FILE* stream);
int ChoosePipe(char* str);

#endif
```

parent.h

```
#ifndef PARENT_H
#define PARENT_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <semaphore.h>
#include <stdbool.h>

void ParentRoutine(FILE* input);

#endif
```

0.0.1 utils.c

```
#include "utils.h"

char* ReadString(FILE* stream)
{
    if(feof(stream)) {
        return NULL;
    }

    const int chunkSize = 256;
    char* buffer = (char*)malloc(chunkSize);
    int bufferSize = chunkSize;

    if(buffer == NULL)
    {
        printf("Couldn't allocate buffer");
        exit(EXIT_FAILURE);
    }

    int readChar;
    int idx = 0;
```

```
    while ((readChar = getc(stream)) != EOF)
    {
        buffer[idx++] = readChar;

        if (idx == bufferSize)
        {
            buffer = realloc(buffer, bufferSize + chunkSize);
            bufferSize += chunkSize;
        }

        if (readChar == '\n') {
            break;
        }
    }

    buffer[idx] = '\0';

    return buffer;
}

int ChoosePipe(char* str)
{
    char* vowels = {"AEIOUYaeiouy"};
    int vowelsCnt = 0;

    char* consonants = {
        "BCDFGHJKLMNPQRSTVWXYZbcdfghjklmnpqrstvwxyz"
    };
    int consonantsCnt = 0;

    for (int i = 0; i < (int)strlen(str); ++i)
    {
        int isVowel = 0;

        for (int j = 0; j < (int)strlen(vowels); ++j){
            if (str[i] == vowels[j])
            {
                ++vowelsCnt;
                isVowel = 1;
                break;
            }
```

```
        }

        if (isVowel){
            continue;
        }

        for (int j = 0; j < (int)strlen(consonants); ++j){
            if (str[i] == consonants[j])
            {
                ++consonantsCnt;
                break;
            }
        }
    }

    return vowelsCnt > consonantsCnt;
}
```

0.0.2 parent.c

```
#include "parent.h"
#include "utils.h"
#include <semaphore.h>

void ParentRoutine(FILE* fin)
{
    char* fileName1 = ReadString(fin);
    char* fileName2 = ReadString(fin);

    fileName1[strlen(fileName1) - 1] = '\0';
    fileName2[strlen(fileName2) - 1] = '\0';

    unlink(fileName1);
    unlink(fileName2);

    pid_t outputFile1, outputFile2;

    if ((outputFile1 = open(fileName1, O_WRONLY | O_CREAT, S_IRWXU)) < 0)
    {
        perror("opening output file 1 error )");
        exit(EXIT_FAILURE);
    }
```

```
if ((outputFile2 = open(fileName2, O_WRONLY | O_CREAT, S_IRWXU)) < 0)
{
    perror("opening output file 2 error )");
    exit(EXIT_FAILURE);
}

free(fileName1);
free(fileName2);

const int mapSize = 128;

void* map1 = mmap(0, mapSize, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_AN
void* map2 = mmap(0, mapSize, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_AN

    sem_t* sem1 = sem_open("semaphore1", O_CREAT, S_IRUSR | S_IWUSR, 0);
    sem_t* sem2 = sem_open("semaphore2", O_CREAT, S_IRUSR | S_IWUSR, 0);

const char* vowels = {"AEIOUYaeiouy"};

pid_t pid1 = fork();
pid_t pid2 = 1;

if (pid1 > 0){
    pid2 = fork();
}

if (pid1 < 0 || pid2 < 0)
{
    perror("Creating process error )");
    exit(EXIT_FAILURE);
}

if (pid1 == 0)
{
    if (dup2(outputFile1, 1) < 0)
    {
        perror("dup2() error )");
        exit(EXIT_FAILURE);
    }

    char* str = (char*)malloc(mapSize);
```

```
            sem_wait(sem1);
    msync(map1, mapSize, MS_SYNC);
    memcpy(str, map1, mapSize);

    for (int i = 0; i < (int)strlen(str); ++i)
    {
        int isVowel = 0;

        for (int j = 0; j < (int)strlen(vowels); ++j){
            if (str[i] == vowels[j])
            {
                isVowel = 1;
                break;
            }
        }

        if (isVowel == 0) {
            printf("%c", str[i]);
        }
    }
}
else if (pid2 == 0)
{
    if (dup2(outputFile2, 1) < 0)
    {
        perror("dup2() error )");
        exit(EXIT_FAILURE);
    }

    char* str = (char*)malloc(mapSize);

            sem_wait(sem2);
    msync(map1, mapSize, MS_SYNC | MS_INVALIDATE);
    memcpy(str, map2, mapSize);

    for (int i = 0; i < (int)strlen(str); ++i)
    {
        int isVowel = 0;

        for (int j = 0; j < (int)strlen(vowels); ++j){
            if (str[i] == vowels[j])
            {
```

```
                isVowel = 1;
                break;
            }
        }

        if (isVowel == 0) {
            printf("%c", str[i]);
        }
    }
}
else
{
    char* strInput = NULL;

    const int chunkSize = 16;

    int bufferSize1 = chunkSize;
    int freeSpace1 = bufferSize1;

    char* buffer1 = (char*)malloc(chunkSize);

    int bufferSize2 = chunkSize;
    int freeSpace2 = bufferSize2;

    char* buffer2 = (char*)malloc(chunkSize);

    while ((strInput = ReadString(fin)) != NULL)
    {
        if (ChoosePipe(strInput))
        {
            while (freeSpace1 < (int)strlen(strInput))
            {
                buffer1 = (char*)realloc(buffer1, bufferSize1 + chunkSize

                bufferSize1 += chunkSize;
                freeSpace1 += chunkSize;
            }

            strcat(buffer1, strInput);
            freeSpace1 -= strlen(strInput);

            free(strInput);
```

```
            }
            else
            {
                while (freeSpace2 < (int)strlen(strInput))
                {
                    buffer2 = (char*)realloc(buffer2, bufferSize2 + chunkSize

                    bufferSize2 += chunkSize;
                    freeSpace2 += chunkSize;
                }

                strcat(buffer2, strInput);
                freeSpace2 -= strlen(strInput);

                free(strInput);
            }
        }

        memcpy(map1, buffer1, strlen(buffer1));
        msync(map1, mapSize, MS_SYNC | MS_INVALIDATE);
                sem_post(sem1);

        memcpy(map2, buffer2, strlen(buffer2));
        msync(map2, mapSize, MS_SYNC | MS_INVALIDATE);
                sem_post(sem2);

                wait(&pid1);
                wait(&pid2);
    }

    munmap(map1, mapSize);
    munmap(map2, mapSize);

    sem_close(sem1);
    sem_close(sem2);
        sem_unlink("semaphore1");
        sem_unlink("semaphore2");
}
```

Выводы

В процессе выполнения лабораторной работы были приобретены навыки реализации обмена информацией с помощью файлов, находящихся в общей памяти процессов.